Analysis of the RSA Encryption Algorithm

Betty Huang

Computer Systems Lab 2009-2010

Abstract

The RSA encryption algorithm is commonly used in public security due to the asymmetric nature of the cipher. The procedure is deceptively simple, though; given two random (large) prime numbers p and q, of which n = pq, and message m, the encrypted text is defined as c = me (mod n). E is some number that is coprime to the totient(n). The public key (n, e), however, makes it difficult for the user to find the private key (n, d), due to the fact that given only n, it is extremely difficult to find the prime factors p and q. The fastest methods currently have O(sqrt(n)) complexity, but require expensive resources and technology (Kaliski). The aim of this paper is to analyze the factorization of n through various popular methods.

Introduction

The RSA algorithm is heavily based on mathematical concepts, utilizing Euler's totient function and prime factorization, and the modulo function. In general, there is a public key function defined as (n, e) and a private key function defined as (n, d). The algorithm statement is given c = m^e % n, where c is the ciphertext and m is the integer representation of the original message, m = c^d % n, for some d. First, note that the totient function returns the number of integers that are coprime to the input, is used to calculate d. Note that with any prime number p, the totient function returns p-1:

p = 5
totient(p) = 4

Since we are looking for n=pq, where p and q are prime numbers, the mathematical multiplication property allows us to calculate totient(n).
p = 5, q = 11
n = pq = 5*11 =55
totient(n) = (p-1)(q-1) = (5-1)(11-1) = 4(10) = 40

Now, we are interested in any positive number e that is coprime to totient(n). In this case, assume e = 3. Now, we want to find d such that de = 1 (mod totient (n)). Since we are guaranteed that a modular multiplicative inverse exists by specifying e such that e is coprime to totient n, we can find d by calculating the modular multiplicative inverse of e modulo totient(n):
3d = 1 (mod 40)
1 (mod 40) = 41, 81, 81+ 40 ...for simplicity, we use 81, because it divides evenly into 3
3d = 81
D = 9

Now, in this example, our public key is (55, 3), and our private key is (55, 9). Now, assume message m = 32 (1 < m < n). Then c= 32^3 % 55 = 43

We could find m by using the decrypt function (m = c^d % n): 43^9 % 55.

Discussion

Following the implementation of the RSA encryption algorithm, I focused on computational techniques on factoring. The majority of the security that follows from the RSA algorithm comes from the age-old problem of factoring. There are various prime factorization methods available, and as time passes, they become more sophisticated in nature and computational power. Following the discussion of the significance of these factorization techniques, I analyzed several algorithms that were within the computational confines of an average processor.

Trial Division
Given the composite--which a number that is divisible-- 225 (a "powerful number), we recognize that 5 divides evenly into 225, leaving us with a remainder of 45, which is divisible by 5, leaving 9. From there, we see that the prime factors of 225 are 3, 3, 5, and 5. The unique property of prime factorization is that there is only one such combination of factors that form a particular composite (the RSA algorithm exploits this property by multiplying two prime numbers together, which means that two prime numbers p and q are the only two possible factors of the composite n).

Fermat's Theorem
If a number n can be expressed into a^2 – b^2, thus the number can be factored into (a+b)(a-b). Coding this algorithm was fairly simple; I would attempt to find an a such that a^2 -N = b^2. This algorithm is only efficient for certain numbers, such as 8051 (8100 – 49).

```
    given n = ab, find a and b.  to do this, we try to represent n as
    the difference of two perfect squares.  for all i > 0, x = sqrt(n) + i.
    calculate x^2 - n for all i until x^2 - n = y^2, a perfect square.
        x^2 - n = y^2,  thus
        n = x^2 - y^2
        n = (x - y)(x + y)
        a = x - y
        b = x + y
```

Pollard's Rho Algorithm
In 1975, John M. Pollard proposed a very efficient Monte Carlo algorithm for factoring, now known as Pollard's ρ method. The algorithm is substantially faster than trial division for finding a small non-trivial factor of a large integer N , if such exists.

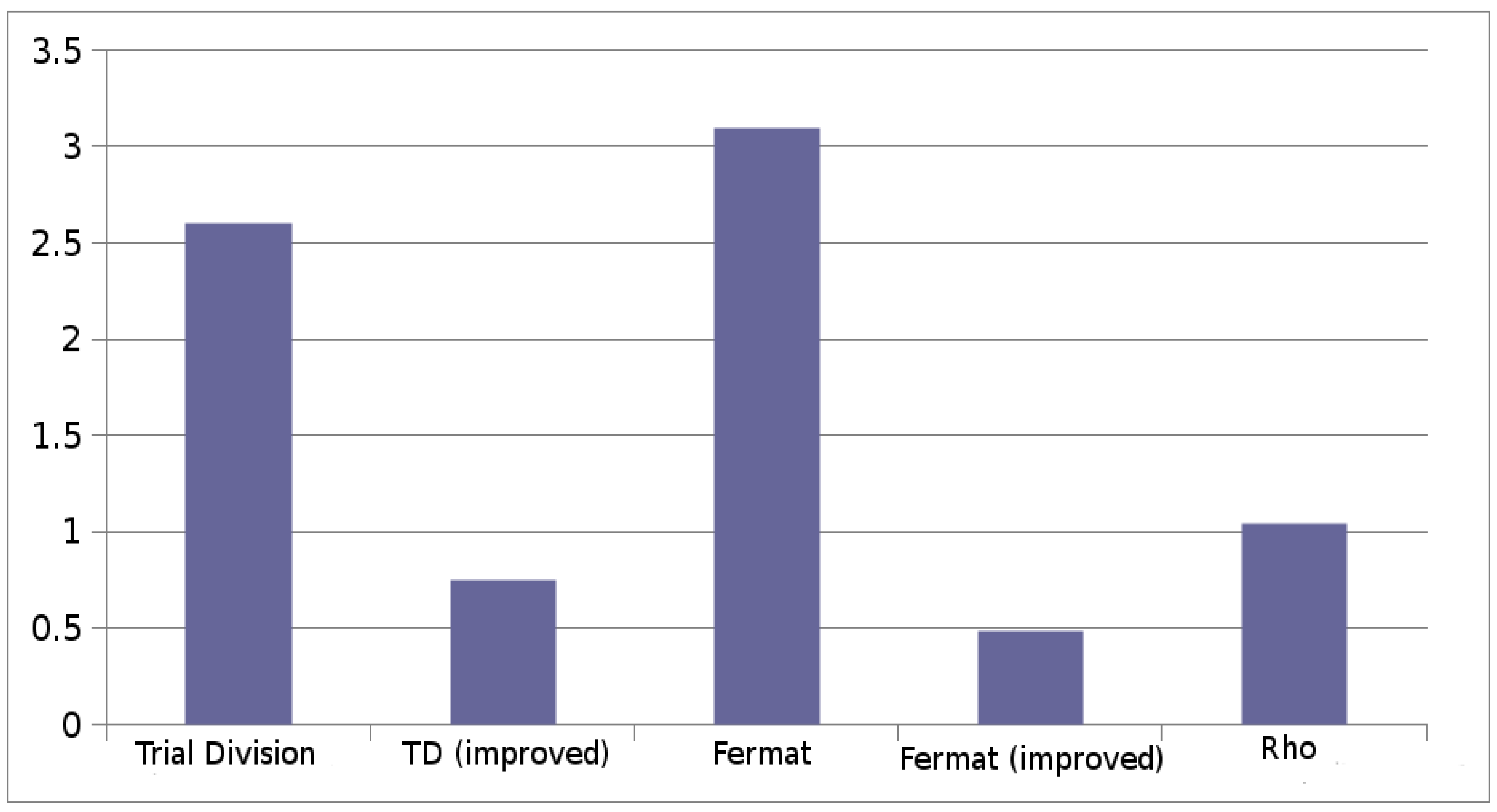
The basic component of Pollard's ρ algorithm is a sequence of pseudo-random integers constructed in the following way:

$$x0 = \mathrm{random}(0, N - 1)$$

$$xi = f(xi - 1) \pmod{N}, i = 1, 2, \cdots$$

where f is a polynomial with integer coefficients, eg. in most practical implementations of the algorithm f (x) = x2 + α, α = 0, − 2.

Results


Trial Division
TD (improved)
Fermat
Fermat (improved)
Rho